Microcontroller Technical Information

CP(K), O

32-Bit Microcontroller V850E/MS2 Usage Restrictions	Document No.	ZBG-CC-07-0006	1/2
	Date issued	March 1, 2007	
	Issued by	2nd Solution Group Multipurpose Microcomputer Systems Division 4th Systems Operations Unit NEC Electronics Corporation	
Related documents V850E/MS2 Hardware User's Manual: U14985E V850E/MS1, V850E/MS2 Architecture User's Manual: U12197E μPD703130 Data sheet: U15390E	Notification classification	√	Usage restriction
			Upgrade
			Document modification
			Other notification

1. Affected products

V850E/MS2

μPD703130: ROMless version

2. Restriction details

A new restriction (No. 8) has been added.

- No. 8 Restriction on *callt* instruction

3. Workaround

The following workaround is available for this restriction. See the attachment for details.

- No. 8 When executing the *callt* instruction, set the base address (CTBP register value) in the internal memory area or do not use the *callt* instruction.

4. Modification schedule

Device modification for restrictions is not planned. Regard the restriction added in this document as a usage restriction.

5. List of restrictions

The restriction history and detailed information is described in the attachment.

6. Document revision history

32-Bit Microcontroller V850E/MS2 Usage Restrictions

Document Number	Date Issued	Description
SBG-DT-0034	November 12, 2001	No. 1, No. 2
SBG-DT-0079	April 16, 2002	No. 1 (modification), No. 3
SBG-DT-0118	July 29, 2002	No. 4, No. 5
SBG-DT-03-0112	March 25, 2003	No. 6
ZBG-CC-04-0005	May 27, 2004	No. 7
ZBG-CC-07-0006 (latest version)	March 1, 2007	No. 8

List of Restrictions in V850E/MS2

1. Product Version

μPD703130: Rank K

* The rank is indicated by the letter appearing as the 5th digit from the left in the lot number marked on each product.

2. Product History

μPD703130

No.	Bugs	Rank
		K
1	Restriction when *sld* instructions are executed consecutively	Δ
2	Restriction when a *bcond* instruction immediately after *sst* instruction is executed	Δ
3	Restriction on A/D converter	Δ
4	Bug whereby DMA transfer is forcibly suspended by NMI	Δ
5	Bug in program execution and DMA transfer in internal RAM	Δ
6	Bug related to conflict between DMA transfer to the internal RAM and instruction executions in the internal RAM	Δ
7	Bug whereby Illegal DMAAK output occurs due to conflict between EDO DRAM access and refresh	Δ
8	Restriction on *callt* instruction	Δ

√: Bug does not occur, Δ: Bug will also apply in future, ×: Bug occurs

3. Details of Usage Restrictions

No. 1 Restriction when *sld* instructions are executed consecutively

[Description]

When interrupt servicing (including NMI servicing) may occur during execution of the *sld* instruction to read data from the external memory space, the data to be loaded may not be transferred to the register.

<Details of bug operation>

When the *sld*-dedicated address calculation resource that is designed for high-speed operation is empty, an *sld* instruction completes address calculation at the ID stage, skips the EX stage and issues a bus cycle at the MEM stage.

If the target of the *sld* instruction is the external memory, issuance of the bus cycle cannot be canceled when an interrupt (including an NMI) is acknowledged at the ID (a) stage. Therefore, the CPU of the V850E/MS2 is designed to skip the WB stage after the MEM stage (read cycle) is issued as a dummy read cycle so that the instruction processing can be canceled.

However, if the WB stage of instruction (1) is delayed (b) when the processing of instruction (2) is canceled, the MEM stage data that is latched by instruction (1) may be overwritten by the dummy read data of instruction (2) at the MEM stage, causing the bug.

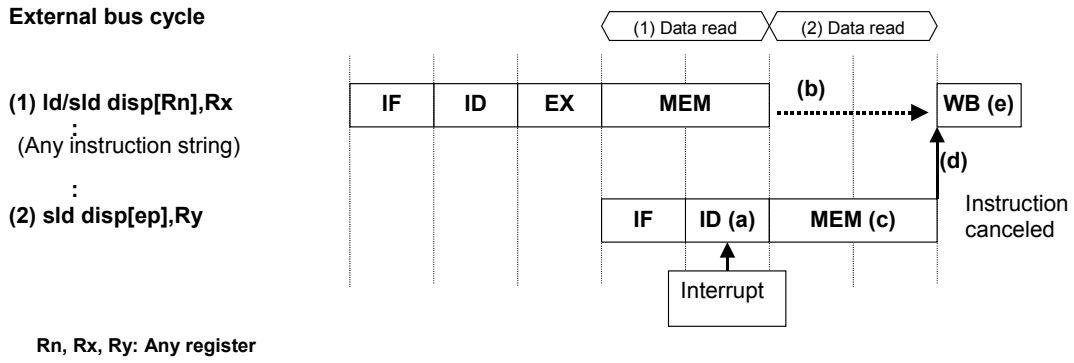


Figure 1-1. Bug Occurrence Mechanism in sld Instruction

The mechanism of this bug is shown below.

(a) An interrupt (including an NMI) is acknowledged at the ID (a) stage of instruction (2).

↓

(b) The preceding instruction (1) shifts to the MEM stage and shifting to the WB stage is delayed due to a WB resource access wait.

↓

(c) Since the issuance of the bus cycle cannot be canceled if the target of the *sld* instruction is the external memory, the MEM stage (c) of instruction (2) is issued as a dummy read cycle.

↓

(d) When the issuance of the bus cycle is canceled by skipping the WB stage of instruction (2), the MEM stage data that is latched by instruction (1) may be overwritten by the dummy read data of instruction (2) at the MEM stage.

↓

(e) The overwritten data is used for instruction (1) of the WB stage (e), causing an erroneous write to the Rx register.

[Conditions under which this bug does not occur]

This bug does not occur when any of the following conditions is satisfied.

(1) When the target of the *sld* instruction is the on-chip memory (including internal RAM)

(2) When interrupts are disabled before/after the *sld* instruction, and no NMI is used

[Workaround]

<Assembler>

Change all the *sld* instructions to the *ld* instruction.

<NEC Electronics compiler>

Do not assign the data that is specified to be assigned to the *tidata* section to a section. Or, change the assignment from the *tidata* section to any other section such as a *sidata* section.

By implementing this workaround, code that does not use the *sld* instruction is created.

<GHS compiler>

This bug can be avoided by implementing the following two workarounds so that the compiler does not output repeated *sld* instructions.

(1) Specify the -Z1412 option at compilation.

(When using the -OS option, also use the -Z1412 and *-inline_prologue* options.)

(2) Avoid using a TDA (Tiny Data Area) function *pragma*.

(When using the TDA area, specify the *-notda* option to make the definition of the TDA area invalid during compilation, or delete all the definitions of the TDA area from the source code.)

<OS (RX850, RX850 Pro)>

This bug does not apply in either of the following cases.

(1) The stack area is set in the internal RAM only.

(2) No NMI is used.

Contact an NEC Electronics sales representative if neither of the above apply.

<Middleware (MH, MR, MMR, JBIG, JPEG, handwriting recognition, Text To Speech, US file, CF driver)>

These middleware products are not affected by this bug.

No. 2 Restriction when a *bcond* instruction is executed immediately after an *sst* instruction

[Description]

Either of the following bugs occurs when the access target of the sst/st instruction (<1>) is the external memory and the *sst* instruction (<2>) is followed by a *bcond* instruction (<3>).

- One word (32 bits) of an instruction code immediately after branching by the *bcond* instruction will not be executed (skipped) and the program counter value becomes incorrect.
- Among the 48-bit instruction code immediately after branching by the *bcond* instruction, any 16 bits will become 0.

As a result, the program following the bug occurrence may not operate correctly. The following shows an example in which this bug occurs. Moreover, this bug operation can also occur at an instruction fetch from the on-chip memory (including RAM) or the external memory.

(1) *sst*/*st* instruction (access to the external memory):

Any instruction strings other than *sst*/*st* instruction (1 or more)

(2) *sst* instruction

(3) *bcond* (bc, be, bge, bgt, bh, bl, ble, blt, bn, bnc, bne, bnh,b nl, bnv, bnz, bp, br, bsa, bv, bz) instruction

- Example of bug operation in instruction at branch address

(1) Among 48-bit instruction code, 16 bits become 0

Correct operation			Bug operation	
PC value	Instruction flow to be executed		PC value	Instruction flow to be executed
0000002A	sst.b r0, 0x0[ep]		0000002A	sst.b r0, 0x0[ep]
0000002C	br 0x40		0000002C	br 0x40
00000040	movhi 0x1010, r0, r6		00000040	movhi 0x0, r0, r6
Instruction code (40361010)		→	Instruction code (40360000)	
00000044	movea 0x1010, r0, r6		00000044	movea 0x1010, r0, r6

(2) One word (32 bits) of an instruction code will not be executed (skipped) and the program counter value becomes incorrect

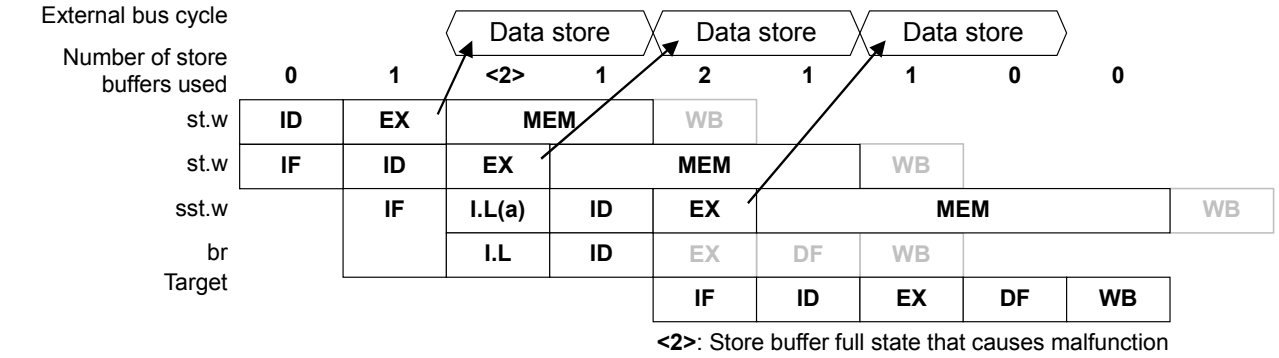
Correct operation		Bug operation	
PC value	Instruction flow to be executed	PC value	Instruction flow to be executed
0000002A	sst.b r0, 0x0[ep]	0000002A	sst.b r0, 0x0[ep]
0000002C	br 0x4e	0000002C	br 0x4e
0000004E	movhi 0x1010, r0, r6	(movhi 0x1010, r0, r6 instruction is not executed)	
00000052	movea 0x1010, r0, r6	0000004E	movea 0x1010, r0, r6 (PC value becomes incorrect)

This bug operation occurs in a pipeline with the following status.

The CPU of the V850E/MS2 has a two-stage store buffer that temporarily stores write data to the memory (**Figure 2-3**). Even if an *sst*/*st* instruction is executed while the external access is being performed, the subsequent instruction can be executed without waiting for the end of the bus cycle as long as the two-stage store buffer has a vacant space. When there is no vacant space in this buffer, the *sst*/*st* instruction is kept waiting at the ID stage until the buffer is released (**Figure 2-2 (a)**).
Usually an *sst* instruction and *bcond* instruction are issued simultaneously. Therefore, if the *bcond* instruction follows the *sst* instruction waiting for the store buffer release, the *bcond* instruction should wait for the buffer release together with the *sst* instruction. Under certain conditions[Note], however, the *bcond* instruction shifts to the ID stage without waiting with the *sst* instruction (**Figure 2-2 (b)**). Due to this malfunction, the fetch operation by the instruction at the branch address (instruction of the target in the figure) becomes abnormal, and cannot operate correctly.

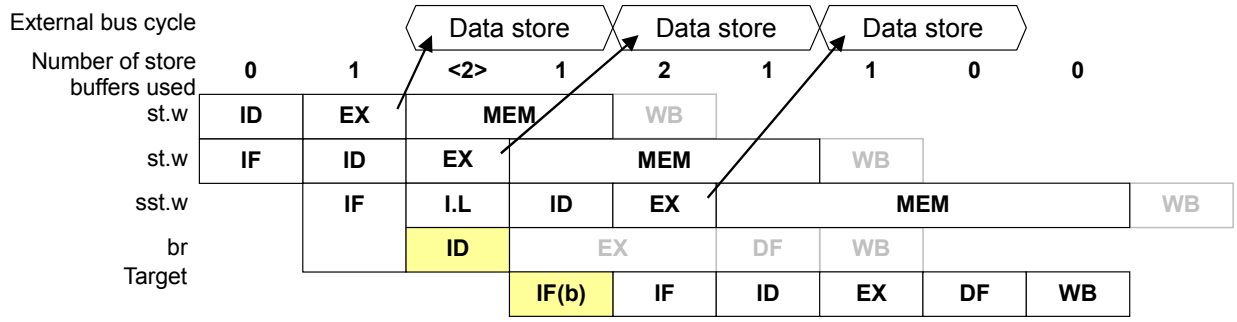
Note Even if the *sst* instruction and *bcond* instruction are executed successively, the bug may not occur depending on the issuance order of the instruction fetch bus cycle and data access bus cycle.

Figure 2-1. Example of Correct Operation



<2>: Store buffer full state that causes malfunction

Figure 2-2. When This Bug Occurs



IF: Instruction fetch
ID: Instruction decode
EX: Execution of the decoded instruction
MEM: Access to the execution target memory
WB: Write the execution result to the register
I.L: Idle state automatically inserted due to a hazard

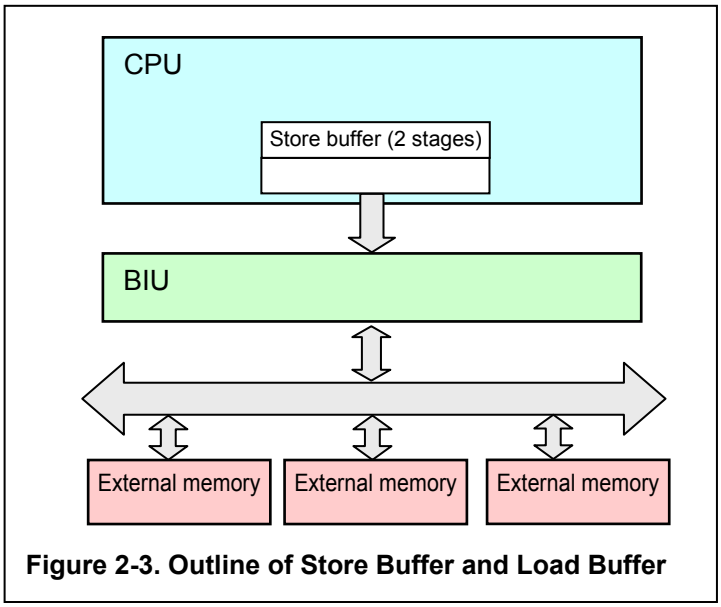


Figure 2-3. Outline of Store Buffer and Load Buffer

[Workaround]

Implement either of the following workarounds by software.

- Replace the *sst* instruction immediately before the *bcond* instruction with the *st* instruction.
- Insert a *nop* instruction between the *sst* instruction and the subsequent *bcond* instruction.

<NEC Electronics compiler>

A corrected version (V2.41) is scheduled for release in mid-November. In V2.41, this bug can be avoided by using the workaround option.

Specify the following option during compilation.

(1) Workaround options in ca850: -Wa, -p

(2) Workaround option in as850: -p

Remark This schedule is subject to change without notice. Contact an NEC Electronics sales representative for the detailed release schedule of V2.41.

<GHS compiler>

This bug can be avoided by implementing the following two workarounds so that the compiler does not output the *sst* instruction.

(1) Specify the -Z1412 option at compilation.

(When using the -OS option, also use the -Z1412 and -*inline_prologue* options.)

(2) Avoid using a TDA (Tiny Data Area) function *pragma*.

(When using the TDA area, specify the -*notda* option to make the definition of the TDA area invalid during compilation, or delete all the definitions of the TDA area from the source code.)

Remark The workaround using the GHS compiler is the same as that in No. 1 Restriction when *sld* instructions are executed consecutively.

<OS (RX850, RX850 Pro)>

These operating systems are not affected by this bug.

<Middleware (MH, MR, MMR, JBIG, JPEG, handwriting recognition, Text To Speech, US file, CF driver)>

These middleware products are not affected by this bug.

No. 3 Restriction on A/D converter

[Description]

This bug occurs when the timer 1 trigger mode of the A/D converter is used (this bug does not occur when the A/D trigger mode or timer 4 trigger mode is used).

Originally, only INTCC110 input can be the trigger to start the A/D converter in the timer 1 trigger mode, and ADTRG pin input in the external trigger mode, therefore the interrupt sources listed below should be ignored. However, if one of the interrupt sources listed in Table 3-1 occurs immediately before the end of A/D conversion (<1> in the Figure 3-1; 2 internal system clocks), it is mistakenly judged as the A/D conversion start trigger. As a result, A/D conversion is started again after the A/D conversion end interrupt (INTAD) is issued. The first A/D conversion ends correctly and the result is stored in the ADCRn register (this value can be read during the second conversion).

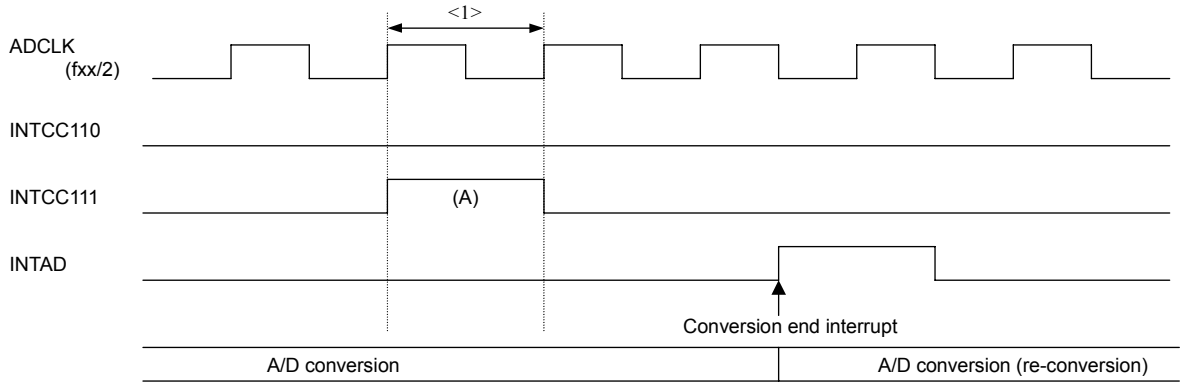
The restarted A/D converter performs the conversion operation correctly, issues the A/D conversion end interrupt (INTAD) and stops. The result is overwritten to the ADCRn register.

Table 3-1. Interrupt Source That Can Trigger A/D Conversion

Timer match interrupt	INTCC111
	INTCC112
	INTCC113
External pin interrupt [Note]	INTP111
	INTP112
	INTP113

Note The external interrupt signal that functions alternately as the external capture trigger input of timer 1 (channel 1) can also be a trigger for re-conversion. In the case of an external interrupt input, this bug occurs when a valid edge is input before the timing of (A) in Figure 3-1 by the noise eliminator (digital noise elimination ($2 \times$ fxx to $3 \times$ fxx ns)).

Figure 3-1. Example of Timing at Which This Bug Occurs



fxx: Internal system clock

[Condition under which this bug does not occur]

This bug does not occur when the A/D trigger mode or timer 4 trigger mode is used.

The condition under which this bug does not occur in the timer 1 trigger mode is shown below.

- The compare match interrupt (INTCC111/INTCC112/INTCC113) of timer 1 (channel 1) does not occur during A/D conversion, and the external interrupt signal (INTCC111/INTCC112/INTCC113) is not input during A/D conversion.

[Workaround]

Since the conversion result is correct even when this bug occurs, the affect of this bug is small when obtaining the latest conversion value. If the re-conversion causes any problems, start A/D conversion in the A/D trigger mode by setting the CE bit of the ADM0 register to 1 in the interrupt service routine of the timer match interrupt.

No. 4 Bug whereby DMA transfer is forcibly suspended by NMI

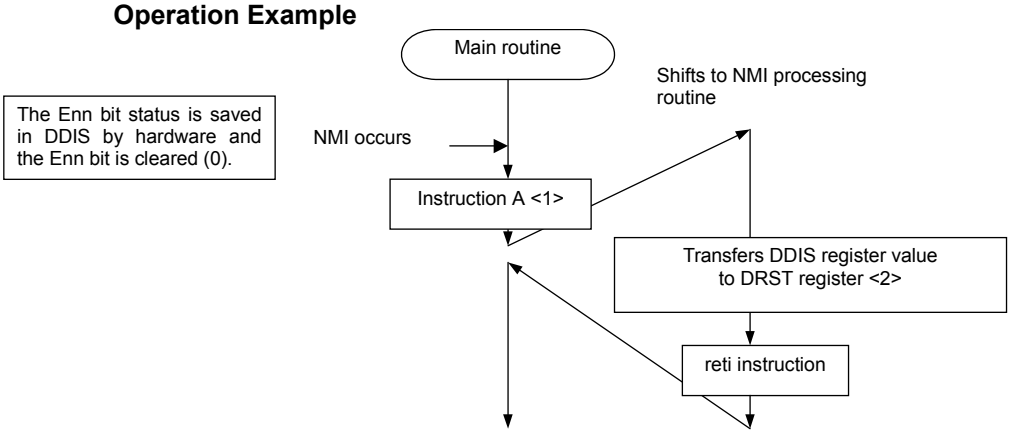
[Description]

DMA transfer is forcibly suspended by an NMI input during a DMA transfer. At this time, the DMA controller saves the status of the Enn bit (bit 0 of the DCHCn register) of all the DMA channels to the DDIS register, clears (0) the Enn bit, and disables the DMA transfer. In addition, when an NMI interrupt is acknowledged in the CPU pipeline operation, an instruction that has already been fetched is executed (1 instruction max.).

If this instruction is the one used to manipulate the Enn bit, the contents of the DDIS register are transferred to the DRST register in the NMI processing routine in order to resume the suspended DMA transfer. Therefore, the Enn bit setting immediately after the NMI input is not reflected.

As a result, the DMA transfer suspended by the NMI input cannot be restored as expected regardless of whether DMA transfer has been disabled or enabled.

Operation Example


Main routine
Shifts to NMI processing routine
The Enn bit status is saved in DDIS by hardware and the Enn bit is cleared (0).
NMI occurs
Instruction A <1>
Transfers DDIS register value to DRST register <2>
reti instruction

When the Enn bit of the DCHCn register is set to disable or enable DMA using instruction A, the disable/enable status of each DMA channel is inconsistent in <1> and <2>. As a result, the normally suspended DMA transfer cannot be restored correctly.

Furthermore, when the Enn bit is set (1) using instruction A while the software trigger bit (STGn) of the DCHCn register is set (1), DMA transfer is performed even in the NMI processing routine.

[Condition under which this bug does not occur]

There is no problem if an NMI is not used.

In addition, there is no problem if the system does not assume to resume the DMA transfer forcibly suspended by the NMI.

[Workaround]

Initialize the DMA transfer forcibly suspended by the NMI and then execute it again.

No. 5 Bug in program execution and DMA transfer in internal RAM

[Description]

When a DMA transfer for the internal RAM and a bit manipulation instruction (SET1, CLR1, or NOT1) allocated in the internal RAM or a data access instruction for a misaligned address are executed simultaneously, the CPU may deadlock due to conflict between the internal bus operations. At this time, only a reset can be acknowledged. (An NMI or interrupt cannot be acknowledged.)

[Condition under which this bug does not occur]

This bug does not occur if no instruction is executed in the internal RAM, or no DMA transfer is performed on the internal RAM.

[Workaround]

Implement either of the following workarounds.

- Do not perform a DMA transfer for the internal RAM when an instruction allocated in the internal RAM is being executed.
- Do not execute an instruction allocated in the internal RAM when a DMA transfer for the internal RAM is being performed.

No. 6 Bug related to conflict between DMA transfer to the internal RAM and instruction executions in the internal RAM

[Description]

When executing a program in the internal RAM, if a conflict occurs between a branch from the internal RAM to the external memory triggered by a branch instruction or interrupt and DMA transfer to the internal RAM, one or two instructions in the external memory are fetched by mistake, which may cause malfunction.

(1) When the following two conditions are satisfied at the same time, the first one or two instructions in the external memory to which the program jumps may be fetched by mistake.

Condition 1: A program is executed in the internal RAM, and a branch to the external memory by a branch instruction (Bcond[Note 1], JR, JARL, JMP, or RETI) is executed.

Condition 2: DMA transfer to the internal RAM[Note 2] is executed.

(2) When the following two conditions are satisfied at the same time, the first one or two instructions in the interrupt handler in the external memory may be fetched by mistake.

Condition 1: A program is executed in the internal RAM, and a branch to the interrupt handler (external memory) by an interrupt (including NMI) is executed.

Condition 2: DMA transfer to the internal RAM[Note 2] is executed.

Note 1: Bcond instructions
[BGT, BGE, BLT, BLE, BH, BNL, BL, BNH, BE, BNE, BV, BNV, BN, BP, BC, BNC, BZ, BNZ, BR, BSA]

Note 2: When the internal RAM is the transfer destination or when the internal RAM is the transfer source

[Conditions under which this bug does not occur]

This bug does not occur when any of the following conditions is satisfied.

(1) The program is not executed in the internal RAM.

(2) The DMA transfer does not target the internal RAM[Note 2].

Note 2: When the internal RAM is the transfer destination or when the internal RAM is the transfer source

[Workaround]

Implement either of the following workarounds.

I. Workaround by software

Disable interrupts before implementing the following workaround when a program in the internal RAM is being executed.

Implement workaround II when an NMI is used.

Execute a dummy read to the internal RAM and a NOP instruction immediately before the branch instruction from the internal RAM to external memory. Branch the program using a word-aligned JMP instruction. An example is shown below.

(a) When the program branches at a JMP instruction

Bug Example	Workaround
Program in the internal RAM Any instruction jmp [Rx] Program in the external memory Any instruction	Program in the internal RAM Any instruction **.align 4** (*1) **nop** **ld.b 0xFFFFE000[R0],R0** **nop** jmp [Rx] (*2) Program in the external memory Any instruction

(*1) Additional instructions after the NOP instruction are aligned to 32 bits and allocated

(*2) Register Rx is the same as register Rx in Bug Example

(b) When the program branches at a RETI instruction

Bug Example	Workaround
Program in the internal RAM Any instruction reti Program in the external memory Any instruction	Program in the internal RAM Any instruction **.align 4** (*1) **nop** **ld.b 0xFFFFE000[R0],R0** **nop** reti Program in the external memory Any instruction

(*1) Additional instructions after the NOP instruction are aligned to 32 bits and allocated

(c) When the program branches at a Bcond or JR instruction

Bug Example	Workaround
Program in the internal RAM Any instruction br/jr EXMEM Program in the external memory EXMEM: Any instruction	Program in the internal RAM Any instruction **.align 4** (*1) **movhi hi(EXMEM),R0,Ry** (*3) **movea lo(EXMEM),Ry,Ry** (*3) **nop** **ld.b 0xFFFFE000[R0],R0** (*3) **nop** **jmp [Ry]** Program in the external memory EXMEM: Any instruction

(*1) Additional instructions after the MOVHI instruction are aligned to 32 bits and allocated

(*3) Register Ry is any unused register

(d) When the program branches at a JARL instruction

Bug Example	Workaround	
Program in the internal RAM	Program in the internal RAM	
Any instruction	Any instruction	
jarl EXMEM,Rz	**.align 4**	(*1)
	movhi hi(EXMEM),R0,Rw	(*4)
	movea lo(EXMEM),Rw,Rw	(*4)
	jarl DUMMYLBL,Rz	(*5)(*6)
	DUMMYLBL:	
	add 0x0a,Rz	(*7)
	ld.b 0xFFFFE000[R0],R0	
	nop	
	jmp [Rw]	
Program in the external memory	Program in the external memory	
EXMEM:	EXMEM:	
Any instruction	Any instruction	

(*1) Additional instructions after the MOVHI instruction are aligned to 32 bits and allocated

(*4) Register Rw is any unused register

(*5) Register Rz is the same as register Rz in Bug Example

(*6) JARL instruction displacement is the next add instruction

(*7) Displacement addition to return to the instruction next to the JMP instruction

II. When workaround I cannot be applied

Implement either of the following workarounds to avoid this bug.

(1) Do not execute programs in the internal RAM when DMA transfer for the internal RAM is performed

(2) Do not execute DMA transfer for the internal RAM when a program is executed in the internal RAM

No. 7 Bug whereby Illegal DMAAK output occurs due to conflict between EDO DRAM access and refresh

[Description]

When performing 2-cycle DMA transfer for which an access to EDO DRAM is started in the “TB” state of an on-page access, if the “TB” state of the EDO DRAM cycle and a refresh request conflict, the state of the $\overline{\text{DMAAKm}}$ signal output is either of the following. (DMA transfer is performed normally.)

(1) When an idle state is not inserted (BCC.BCn1 and BCC.BCn0 bits = 00)
The $\overline{\text{DMAAKm}}$ signal output becomes inactive when the read cycle ends.

(2) When an idle state is inserted (BCC.BCn1 and BCC.BCn0 bits = 01)

The $\overline{\text{DMAAK}}$m signal output becomes inactive in the idle state between the read and write cycles, and becomes active again in the next write cycle.

Remark m = 0 to 3, n =0 to 7

[Conditions under which this bug occurs]

This bug occurs when all the following conditions are satisfied.

Condition 1: 2-cycle DMA transfer from external memory to external memory or external I/O.

Condition 2: The transfer destination is EDO DRAM.

Condition 3: No data wait is inserted in EDO DRAM.
(DRCp.DAC1p and DRCp.DAC0p bits = 00)

Condition 4: No CAS precharge wait is inserted in EDO DRAM
(DRCp.CPC1p and DRCp.CPC0p bits = 00)

Condition 5: The refresh function is valid.

Remark p = 0 to 3

[Workaround]

Implement either of the following workarounds so that the EDO DRAM access cycle does not start in the “TB” state.

(1) Insert a data wait in EDO DRAM.
(DRCp.DAC1p and DRCp.DAC0p bits = 01, 10, or 11)

(2) Insert a CAS precharge wait in EDO DRAM.
(DRCp.CPC1p and DRCp.CPC0p bits = 01, 10, or 11)

Remark p = 0 to 3

No. 8 Restriction on *callt* instruction

[Description]

In the instruction sequences shown below, if the base address (CTBP register value) of the *callt* instruction is set in an external memory area and interrupt servicing (including NMI servicing) is executed during execution of the *callt* instruction, the result of the load instruction which has been executed prior may be overwritten by the CTBP value.

Instruction sequences in question:

(1) ld/sld disp[Rn], Rx Rn, Rx: Arbitrary register
: (arbitrary instruction line)

(2) call imm16

[Conditions under which this bug does not occur]

This bug does not occur when one of the following conditions is satisfied.

(1) When the base pointer to the *callt* instruction points to an internal memory (including internal RAM).

(2) When interrupt servicing before and after the *callt* instruction is disabled and NMI is not used.

(3) When the *callt* instruction is not used

[Workaround]

<When using NEC Electronics C compiler CA850 (V2.50 or later)>

Implement either of the following workarounds.

- Prevent the *callt* instruction from being output

  Specify *–Xpro_epi_runtime = off* for the compiler option.

- Assign the base address of the *callt* instruction in the internal RAM.

  The *callt* instruction references the table in the following section.

  .pro_epi_runtime

To allocate this section to the internal RAM, allocate the following part to the internal RAM, in the link directive file.

```
TEXT        : !LOAD ?RX {
              .pro_epi_runtime = $PROGBITS        ?AX  .pro_epi_runtime; ← This part
              .text            = $PROGBITS        ?AX  .text;
};
```

Example When *text* is allocated to the external memory

- Before change

```
TEXT        : !LOAD ?RX V0x100000 {
              .pro_epi_runtime = $PROGBITS        ?AX  .pro_epi_runtime; ← This part
              .text            = $PROGBITS        ?AX  .text;
};
```

- After change

```
PETEXT      : !LOAD ?RX V0x1000 {
              .pro_epi_runtime = $PROGBITS        ?AX  .pro_epi_runtime;
};
TEXT        : !LOAD ?RX V0x100000 {
              .text            = $PROGBITS        ?AX  .text;
};
```

<OS (RX850, RX850Pro, or RX850V4)>

These operating systems are not affected by this case because the *callt* instruction is not used.

<CA850 library “gofast”>

This library is not affected by this case because the *callt* instruction is not used.

<RX-NET850 (Including option protocol), RX-FS850 (V2.00)>

The *callt* instruction is not used in the library. When using the sample driver (for SG-MA1) included with the RX-FS850 (CA850 version), however, change the “Default[Use]” setting for the Prologue Epilogue Runtime drop-down list in the Compiler Options dialog box to “No use”, then re-build it.

<GHS compiler (V3.x series (V3.5 (R6.5.2) or later), V4.x series (V4.0.2 (R7.0.1) or later)>

Implement either of the following workarounds.

- Do not output the callt instruction

  Specify *-no_callt* for the compiler option.

- Assign the base address of the *callt* instruction in the internal RAM.

  The *callt* instruction references the table in the following section.

  .text

  The *callt* instruction table is included in the *indarchl.o* module in *libarch.a*.

To allocate this section to the internal RAM, allocate the following code to the internal RAM, in the link directive file.

```
MEMORY {
                    <snip>
                    iram_memory    :ORIGIN = 0 x 0, LENGTH = 0 x 1000
                    <snip>
}
SECTION {
                    <snip>
                    .newtext: {libarch.a (indarchl. o (.text))} > iram_memory
                    <snip>
}
```

Remark For measures other than the above, consult an NEC Electronics sales representative.